Vaishnav Eega

BT22CSH019

Code :

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <math.h>

*// Node structure for polynomial terms*

*typedef* *struct* Node {

*float* *coeff*;

*int* *exp*;

*struct* Node*\** *next*;

} Node;

*// Function to create a new node*

Node*\** *createNode*(*float* coeff, *int* exp) {

    Node*\** *newNode* *=* (Node*\**)*malloc*(*sizeof*(Node));

*if* (*newNode* *==* *NULL*) {

*printf*("Memory allocation failed\n");

*exit*(1);

    }

*newNode*->*coeff* *=* coeff;

*newNode*->*exp* *=* exp;

*newNode*->*next* *=* *NULL*;

*return* *newNode*;

}

*// Function to insert a term into the polynomial*

*void* *insertTerm*(Node*\*\** header, *float* coeff, *int* exp) {

    Node*\** *newNode* *=* *createNode*(coeff, exp);

*if* (*\**header *==* *NULL*) {

*\**header *=* *createNode*(0, 0); *// Initialize the circular linked list with a header node*

*newNode*->*next* *=* *\**header;

        (*\**header)->*next* *=* *newNode*;

*return*;

    }

    Node*\** *current* *=* *\**header;

*while* (*current*->*next* *!=* *\**header *&&* *current*->*next*->*exp* *>* exp) {

*current* *=* *current*->*next*;

    }

*newNode*->*next* *=* *current*->*next*;

*current*->*next* *=* *newNode*;

}

*// Function to read a polynomial and convert it to circular representation*

Node*\** *Pread*() {

    Node*\** *header* *=* *NULL*;

*int* *numTerms*;

*printf*("Enter the number of terms in the polynomial: ");

*scanf*("%d", *&numTerms*);

*for* (*int* *i* *=* 0; *i* *<* *numTerms*; *++i*) {

*float* *coeff*;

*int* *exp*;

*printf*("Enter coefficient and exponent for term %d: ", *i* *+* 1);

*scanf*("%f %d", *&coeff*, *&exp*);

*insertTerm*(*&header*, *coeff*, *exp*);

    }

*return* *header*;

}

*// Function to output the polynomial in mathematical form*

*void* *Pwrite*(Node*\** header) {

    Node*\** *current* *=* header->*next*;

*while* (*current* *!=* header) {

*printf*("%.2fx^%d", *current*->*coeff*, *current*->*exp*);

*current* *=* *current*->*next*;

*if* (*current* *!=* header) {

*printf*(" + ");

        }

    }

*printf*("\n");

}

*// Function to add two polynomials*

Node*\** *Padd*(Node*\** a, Node*\** b) {

    Node*\** *result* *=* *NULL*;

    Node*\** *currentA* *=* a->*next*;

    Node*\** *currentB* *=* b->*next*;

*while* (*currentA* *!=* a *&&* *currentB* *!=* b) {

*if* (*currentA*->*exp* *>* *currentB*->*exp*) {

*insertTerm*(*&result*, *currentA*->*coeff*, *currentA*->*exp*);

*currentA* *=* *currentA*->*next*;

        } *else* *if* (*currentA*->*exp* *<* *currentB*->*exp*) {

*insertTerm*(*&result*, *currentB*->*coeff*, *currentB*->*exp*);

*currentB* *=* *currentB*->*next*;

        } *else* {

*float* *sumCoeff* *=* *currentA*->*coeff* *+* *currentB*->*coeff*;

*if* (*fabs*(*sumCoeff*) *>* 1*e-*6) {

*insertTerm*(*&result*, *sumCoeff*, *currentA*->*exp*);

            }

*currentA* *=* *currentA*->*next*;

*currentB* *=* *currentB*->*next*;

        }

    }

*while* (*currentA* *!=* a) {

*insertTerm*(*&result*, *currentA*->*coeff*, *currentA*->*exp*);

*currentA* *=* *currentA*->*next*;

    }

*while* (*currentB* *!=* b) {

*insertTerm*(*&result*, *currentB*->*coeff*, *currentB*->*exp*);

*currentB* *=* *currentB*->*next*;

    }

*return* *result*;

}

*// Function to subtract two polynomials*

Node*\** *Psub*(Node*\** a, Node*\** b) {

    Node*\** *result* *=* *NULL*;

    Node*\** *currentA* *=* a->*next*;

    Node*\** *currentB* *=* b->*next*;

*while* (*currentA* *!=* a *&&* *currentB* *!=* b) {

*if* (*currentA*->*exp* *>* *currentB*->*exp*) {

*insertTerm*(*&result*, *currentA*->*coeff*, *currentA*->*exp*);

*currentA* *=* *currentA*->*next*;

        } *else* *if* (*currentA*->*exp* *<* *currentB*->*exp*) {

*insertTerm*(*&result*, *-currentB*->*coeff*, *currentB*->*exp*);

*currentB* *=* *currentB*->*next*;

        } *else* {

*float* *diffCoeff* *=* *currentA*->*coeff* *-* *currentB*->*coeff*;

*if* (*fabs*(*diffCoeff*) *>* 1*e-*6) {

*insertTerm*(*&result*, *diffCoeff*, *currentA*->*exp*);

            }

*currentA* *=* *currentA*->*next*;

*currentB* *=* *currentB*->*next*;

        }

    }

*while* (*currentA* *!=* a) {

*insertTerm*(*&result*, *currentA*->*coeff*, *currentA*->*exp*);

*currentA* *=* *currentA*->*next*;

    }

*while* (*currentB* *!=* b) {

*insertTerm*(*&result*, *-currentB*->*coeff*, *currentB*->*exp*);

*currentB* *=* *currentB*->*next*;

    }

*return* *result*;

}

*// Function to multiply two polynomials*

Node*\** *Pmult*(Node*\** a, Node*\** b) {

    Node*\** *result* *=* *NULL*;

    Node*\** *currentA* *=* a->*next*;

*while* (*currentA* *!=* a) {

        Node*\** *currentB* *=* b->*next*;

*while* (*currentB* *!=* b) {

*insertTerm*(*&result*, *currentA*->*coeff* *\** *currentB*->*coeff*, *currentA*->*exp* *+* *currentB*->*exp*);

*currentB* *=* *currentB*->*next*;

        }

*currentA* *=* *currentA*->*next*;

    }

*return* *result*;

}

*// Function to evaluate the polynomial at a specific point*

*float* *Peval*(Node*\** header, *float* x) {

*float* *result* *=* 0.0;

    Node*\** *current* *=* header->*next*;

*while* (*current* *!=* header) {

*result* *+=* *current*->*coeff* *\** *pow*(x, *current*->*exp*);

*current* *=* *current*->*next*;

    }

*return* *result*;

}

*// Function to erase a term with a specific exponent*

*void* *Pearse*(Node*\*\** header, *int* exp) {

    Node*\** *current* *=* *\**header;

*while* (*current*->*next* *!=* *\**header) {

*if* (*current*->*next*->*exp* *==* exp) {

            Node*\** *temp* *=* *current*->*next*;

*current*->*next* *=* *temp*->*next*;

*free*(*temp*);

*return*;

        }

*current* *=* *current*->*next*;

    }

}

*// Function to free memory occupied by the linked list*

*void* *freeList*(Node*\** header) {

    Node*\** *current* *=* header->*next*;

*while* (*current* *!=* header) {

        Node*\** *temp* *=* *current*;

*current* *=* *current*->*next*;

*free*(*temp*);

    }

*free*(header);

}

*int* *main*() {

    Node*\** *a* *=* *NULL*;

    Node*\** *b* *=* *NULL*;

*printf*("Enter polynomial A:\n");

*a* *=* *Pread*();

*printf*("Enter polynomial B:\n");

*b* *=* *Pread*();

    Node*\** *c* *=* *Padd*(*a*, *b*);

    Node*\** *d* *=* *Psub*(*a*, *b*);

    Node*\** *e* *=* *Pmult*(*a*, *b*);

*printf*("A: ");

*Pwrite*(*a*);

*printf*("B: ");

*Pwrite*(*b*);

*printf*("A + B: ");

*Pwrite*(*c*);

*printf*("A - B: ");

*Pwrite*(*d*);

*printf*("A \* B: ");

*Pwrite*(*e*);

*// Evaluate polynomials at a specific point*

*float* *x*;

*printf*("Enter the value of x to evaluate polynomials: ");

*scanf*("%f", *&x*);

*printf*("A(%f) = %f\n", *x*, *Peval*(*a*, *x*));

*printf*("B(%f) = %f\n", *x*, *Peval*(*b*, *x*));

*// Erase a term from a polynomial*

*int* *exp*;

*printf*("Enter the exponent of the term to erase from polynomial A: ");

*scanf*("%d", *&exp*);

*Pearse*(*&a*, *exp*);

*printf*("A after erasing term with exponent %d: ", *exp*);

*Pwrite*(*a*);

*// Free memory*

*freeList*(*a*);

*freeList*(*b*);

*freeList*(*c*);

*freeList*(*d*);

*freeList*(*e*);

*return* 0;

}

Ouput :

